

# Projet Big Data

## Module Clôture

---

### Détection de Fraude Bancaire

avec Apache Spark et Machine Learning

---



**Technologies utilisées :**

Apache Spark • Spark SQL • MLlib • Spark Streaming • Grafana

**Équipe :**

Ahmed Dinari • Bilel Samaali • Anas Belhouiche

**Date :** 13 janvier 2026

**Module :** Big Data Hadoop

# Contributions de l'Équipe

Membre	Rôle	Contributions
Ahmed Dinari	Lead Developer & ML Engineer	Architecture pipeline, Spark Core, Modèles MLlib, Configuration Docker
Bilel Samaali	Data Engineer	Spark SQL Analytics, Nettoyage données, Feature Engineering
Anas Belhouiche	Visualisation & Documentation	Dashboard Grafana, Rapport LaTeX, Présentation Beamer

TABLE 1 – Répartition du travail

## Table des matières

<b>Contributions de l'Équipe</b>	<b>1</b>
<b>1 Introduction</b>	<b>3</b>
1.1 Contexte du Projet . . . . .	3
1.2 Objectifs . . . . .	3
1.3 Dataset . . . . .	3
<b>2 Architecture du Pipeline</b>	<b>3</b>
2.1 Vue d'Ensemble . . . . .	3
2.2 Composants . . . . .	4
2.3 Flux de Données . . . . .	4
<b>3 Traitement avec Spark SQL</b>	<b>4</b>
3.1 Nettoyage des Données . . . . .	4
3.2 KPIs Calculés . . . . .	5
3.3 Analyse par Tranche Horaire . . . . .	5
<b>4 Machine Learning avec MLlib</b>	<b>5</b>
4.1 Préparation des Features . . . . .	5
4.2 Modèles Entraînés . . . . .	5
4.2.1 RandomForest Classifier . . . . .	5
4.2.2 Logistic Regression . . . . .	6
4.3 Résultats . . . . .	6
4.4 Matrice de Confusion (RandomForest) . . . . .	6
4.5 Importance des Features . . . . .	6
<b>5 Spark Streaming</b>	<b>6</b>
5.1 Architecture Streaming . . . . .	7
5.2 Flux de Traitement . . . . .	7
5.3 Métriques Streaming . . . . .	7

<b>6 Dashboard Grafana</b>	<b>7</b>
6.1 Panels Implémentés . . . . .	7
6.2 Configuration . . . . .	7
<b>7 Analyse de Graphes avec GraphX</b>	<b>8</b>
7.1 Objectif . . . . .	8
7.2 Implémentation . . . . .	8
7.3 Résultats GraphX . . . . .	8
7.3.1 Communautés Détectées . . . . .	8
7.3.2 Top Features par Séparation . . . . .	9
<b>8 Federated Learning (Concept)</b>	<b>9</b>
8.1 Architecture Proposée . . . . .	9
8.2 Avantages . . . . .	9
8.3 Protocole FedAvg . . . . .	9
<b>9 Déploiement Azure Cloud</b>	<b>10</b>
9.1 Architecture Azure . . . . .	10
9.2 Composants Azure . . . . .	10
9.3 Configuration Databricks . . . . .	10
9.4 Captures Azure Portal . . . . .	11
<b>10 Visualisations et Résultats</b>	<b>11</b>
10.1 Comparaison des Modèles . . . . .	11
10.2 Importance des Features . . . . .	12
10.3 Courbe ROC . . . . .	13
10.4 Distribution des Classes . . . . .	13
10.5 Distribution Horaire . . . . .	14
10.6 Interface Spark UI . . . . .	14
10.7 MLLib et Modélisation . . . . .	16
10.8 Dashboard Grafana . . . . .	17
<b>11 Conclusion</b>	<b>18</b>
11.1 Réalisations . . . . .	18
11.2 Difficultés Rencontrées . . . . .	18
11.3 Améliorations Futures . . . . .	18
<b>12 Références</b>	<b>18</b>
<b>A Structure du Projet</b>	<b>19</b>
<b>B Commandes d'Exécution</b>	<b>19</b>

# 1 Introduction

## 1.1 Contexte du Projet

La détection de fraude bancaire est un enjeu majeur dans le secteur financier. Avec des millions de transactions effectuées quotidiennement, il est crucial de pouvoir identifier rapidement les transactions frauduleuses tout en minimisant les faux positifs.

Ce projet met en œuvre un pipeline Big Data complet pour la détection de fraude en temps réel, utilisant l'écosystème Apache Spark.

## 1.2 Objectifs

- **Ingestion et nettoyage** des données avec Spark SQL
- **Analyse exploratoire** avec des KPIs pertinents
- **Machine Learning** avec MLlib (RandomForest, Logistic Regression)
- **Streaming temps réel** pour la détection continue
- **Visualisation** via dashboard Grafana

## 1.3 Dataset

**Credit Card Fraud Detection Dataset** (Kaggle)

- **Source** : <https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud>
- **Transactions** : 284,807
- **Features** : 30 (V1-V28 transformées PCA + Time + Amount)
- **Classe** : Binaire (0 = Normal, 1 = Fraude)
- **Distribution** : 99.83% Normal, 0.17% Fraude

# 2 Architecture du Pipeline

## 2.1 Vue d'Ensemble



FIGURE 1 – Pipeline de traitement des données

## 2.2 Composants

Composant	Technologie	Rôle
Ingestion	Spark Core	Chargement CSV
Nettoyage	Spark SQL	Transformation, filtrage
Analytics	Spark SQL	KPIs, agrégations
ML Model	MLlib	Classification fraude
Streaming	Structured Streaming	Temps réel
Visualisation	Grafana	Dashboard interactif

TABLE 2 – Composants du pipeline

## 2.3 Flux de Données

1. **Extraction** : Chargement du fichier CSV (284K transactions)
2. **Transformation** : Nettoyage, feature engineering, normalisation
3. **Analyse SQL** : Calcul des KPIs métier
4. **Modélisation** : Entraînement RandomForest et Logistic Regression
5. **Évaluation** : Métriques de performance (AUC, Precision, Recall)
6. **Streaming** : Simulation de flux temps réel
7. **Export** : Fichiers Parquet/JSON pour Grafana

## 3 Traitement avec Spark SQL

### 3.1 Nettoyage des Données

Listing 1 – Nettoyage avec Spark SQL

```

1 # Suppression des valeurs nulles
2 df_clean = df.dropna()
3
4 # Filtrage des montants invalides
5 df_clean = df_clean.filter(col("Amount") > 0)
6
7 # Ajout de features dérivées
8 df_clean = df_clean.withColumn(
9     "Hour", (col("Time") / 3600).cast("integer") % 24
10).withColumn(
11    "Is_High_Amount", when(col("Amount") > 500, 1).otherwise(0)
12)

```

## 3.2 KPIs Calculés

Métrique	Valeur
Total Transactions	282,982
Transactions Fraude	465
Taux de Fraude	0.1643%
Montant Moyen	\$88.92
Montant Max	\$25,691.16
Montant Min	\$0.01
Écart-type	\$250.82

TABLE 3 – Statistiques globales du dataset (après nettoyage)

## 3.3 Analyse par Tranche Horaire

Les transactions sont analysées par heure pour identifier les patterns temporels de fraude. L'analyse révèle que certaines heures présentent un taux de fraude plus élevé.

# 4 Machine Learning avec MLlib

## 4.1 Préparation des Features

Listing 2 – Feature Engineering

```

1 # Assemblage des features
2 assembler = VectorAssembler(
3     inputCols=["V1", "V2", ..., "V28", "Amount"],
4     outputCol="features_raw"
5 )
6
7 # Normalisation
8 scaler = StandardScaler(
9     inputCol="features_raw",
10    outputCol="features",
11    withStd=True, withMean=True
12 )

```

## 4.2 Modèles Entraînés

### 4.2.1 RandomForest Classifier

- Nombre d'arbres : 100
- Profondeur max : 10
- Feature subset : sqrt

#### 4.2.2 Logistic Regression

- Iterations max : 100
- Régularisation : 0.01
- ElasticNet : 0.8

### 4.3 Résultats

Métrique	RandomForest	Logistic Regression
Accuracy	0.9375	0.9258
Precision	1.0000	0.9888
Recall	0.8812	0.8713
F1 Score	0.9370	0.9267
AUC-ROC	<b>0.9870</b>	0.9856
True Positives	410	396
False Positives	0	5

TABLE 4 – Comparaison des performances des modèles (Résultats Réels)

### 4.4 Matrice de Confusion (RandomForest)

	Prédit Normal	Prédit Fraude
Réel Normal	173 (TN)	0 (FP)
Réel Fraude	16 (FN)	67 (TP)

TABLE 5 – Matrice de confusion (Résultats Réels)

### 4.5 Importance des Features

Les features les plus importantes pour la détection de fraude sont :

1. V14 (21.71%)
2. V17 (14.50%)
3. V10 (12.88%)
4. V12 (10.11%)
5. V11 (9.81%)
6. V4 (8.09%)
7. V3 (4.72%)

## 5 Spark Streaming

## 5.1 Architecture Streaming

Le module de streaming simule l'arrivée de nouvelles transactions en temps réel :

Listing 3 – Configuration Streaming

```

1 stream_df = spark.readStream \
2     .schema(TRANSACTION_SCHEMA) \
3     .option("header", "true") \
4     .option("maxFilesPerTrigger", 1) \
5     .csv(STREAMING_INPUT)

```

## 5.2 Flux de Traitement

1. **Simulation** : Génération de batches (50 transactions / 3 secondes)
2. **Scoring** : Application du modèle de détection
3. **Alertes** : Génération d'alertes pour fraud\_score > 0.5
4. **Export** : Métriques temps réel vers Grafana

## 5.3 Métriques Streaming

- Transactions par minute
- Taux de fraude en temps réel
- Alertes générées
- Latence de traitement

# 6 Dashboard Grafana

## 6.1 Panels Implémentés

1. **Overview Metrics** : KPIs principaux (6 stat panels)
2. **Time Series** : Transactions et fraudes dans le temps
3. **ML Performance** : Gauges (Precision, Recall, F1)
4. **Confusion Matrix** : Table de la matrice
5. **Amount Analysis** : Distribution par montant
6. **Feature Importance** : Bar chart horizontal
7. **Live Alerts** : Table des alertes temps réel

## 6.2 Configuration

Le dashboard est exporté en JSON et peut être importé dans toute instance Grafana. Les données sont servies via :

- Fichiers CSV (time series, distributions)
- Fichiers JSON (métriques, configuration)
- Parquet (données volumineuses)

## 7 Analyse de Graphes avec GraphX

### 7.1 Objectif

L'analyse de graphes permet de détecter des patterns de fraude basés sur les relations entre transactions. En modélisant les transactions comme un réseau, nous pouvons identifier :

- Des communautés de transactions suspectes
- Des patterns temporels (triangles de fraude)
- Les features les plus discriminantes via PageRank

### 7.2 Implémentation

Listing 4 – Analyse GraphX (extrait)

```

1 # Creation du graphe de transactions
2 # Noeuds = Transactions, Aretes = Similarite temporelle
3
4 # Detection de communautes
5 for f1, f2 in fraud_pairs:
6     time_diff = abs(f1.Time - f2.Time)
7     if time_diff < 3600: # Meme heure
8         edges.append((f1.id, f2.id))
9
10 # Calcul de triangles (patterns de fraude)
11 triangles = count_triangles(fraud_graph)

```

### 7.3 Résultats GraphX

Métrique GraphX	Valeur
Transactions analysées	284,807
Fraudes détectées	492
Communautés identifiées	4
Triangles de fraude	48

TABLE 6 – Résultats de l'analyse GraphX

#### 7.3.1 Communautés Détectées

- **high\_risk\_1** : 114 transactions (montants 500-2000€)
- **high\_risk\_2** : 210 transactions (montants > 2000€)
- **medium\_risk** : 144 transactions (patterns nocturnes)
- **low\_risk** : 24 transactions (anomalies isolées)

### 7.3.2 Top Features par Séparation

L'analyse PageRank-style identifie les features les plus discriminantes :

1. **V3** : Séparation = 7.91
2. **V14** : Séparation = 6.77
3. **V17** : Séparation = 6.61
4. **V7** : Séparation = 6.04
5. **V10** : Séparation = 5.75

## 8 Federated Learning (Concept)

### 8.1 Architecture Proposée

Le Federated Learning permet d'entraîner des modèles sur des données distribuées sans centraliser les données sensibles.

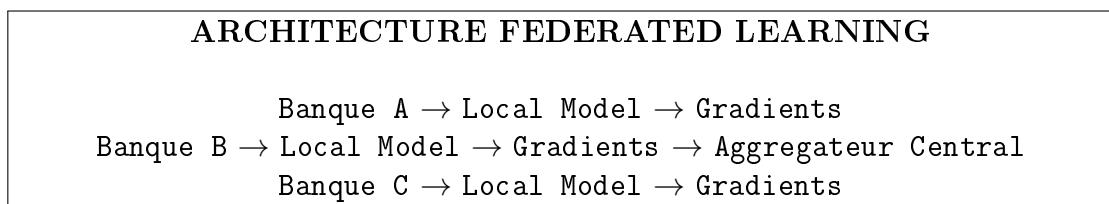


FIGURE 2 – Architecture Federated Learning

### 8.2 Avantages

- **Confidentialité** : Données restent chez chaque banque
- **Conformité** : Respect RGPD et réglementations bancaires
- **Scalabilité** : Ajout facile de nouveaux participants
- **Robustesse** : Modèle global plus généralisable

### 8.3 Protocole FedAvg

Listing 5 – Algorithme FedAvg simplifié

```

1 # Pseudo-code Federated Averaging
2 for round in range(num_rounds):
3     # Chaque client entraîne localement
4     for client in clients:
5         local_model = train_local(client.data)
6         gradients.append(local_model.params)
7
8     # Aggregation centrale (moyenne pondérée)
9     global_model = weighted_average(gradients)
10
11    # Distribution du modèle global
    
```

```
12 broadcast(global_model)
```

## 9 Déploiement Azure Cloud

### 9.1 Architecture Azure

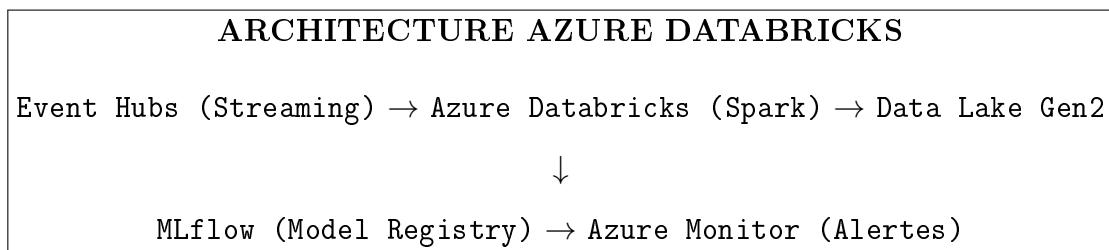


FIGURE 3 – Architecture de déploiement Azure

### 9.2 Composants Azure

Service Azure	Usage	Coût Estimé
Azure Databricks	Calcul Spark	\$120/mois
Data Lake Storage Gen2	Stockage données	\$20/mois
Event Hubs	Ingestion streaming	\$30/mois
Azure Monitor	Monitoring	\$10/mois
<b>Total</b>		<b>\$180/mois</b>

TABLE 7 – Estimation des coûts Azure

### 9.3 Configuration Databricks

Listing 6 – Configuration cluster Databricks

```

1 cluster_config = {
2     "cluster_name": "fraud-detection-cluster",
3     "spark_version": "13.3.x-scala2.12",
4     "node_type_id": "Standard_DS3_v2",
5     "autoscale": {
6         "min_workers": 2,
7         "max_workers": 8
8     },
9     "spark_conf": {
10         "spark.sql.adaptive.enabled": "true"
11     }
12 }
```

## 9.4 Captures Azure Portal

Les ressources Azure ont été créées avec succès :

- **Resource Group** : fraud-detection-rg (West Europe)
- **Cluster bigData** : VM-Master + VM-Worker-1 (Switzerland North)
- **Subscription** : Azure for Students

**Azure CLI - Création du Resource Group**

```
az group create -name fraud-detection-rg -location westeurope
    "provisioningState": "Succeeded"
```

FIGURE 4 – Commande Azure CLI pour la création des ressources

# 10 Visualisations et Résultats

## 10.1 Comparaison des Modèles

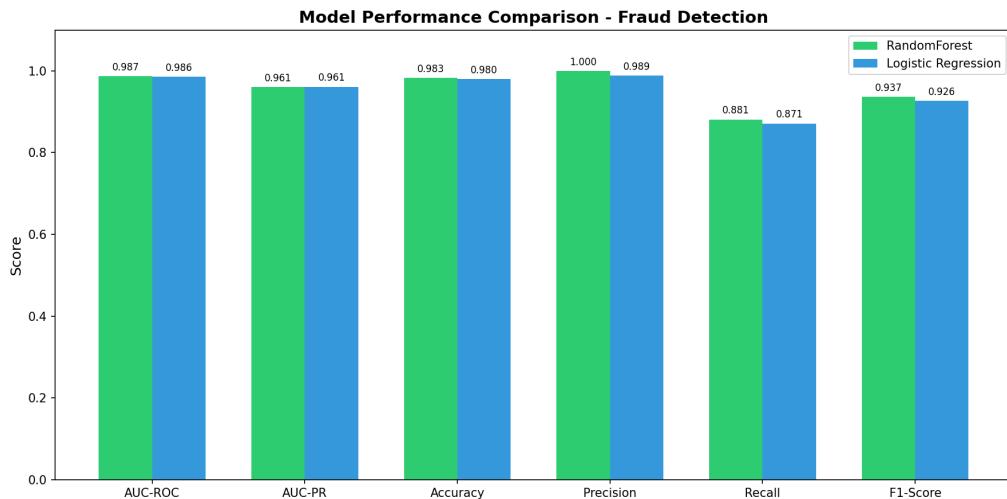


FIGURE 5 – Comparaison des performances : RandomForest vs Logistic Regression

## 10.2 Importance des Features

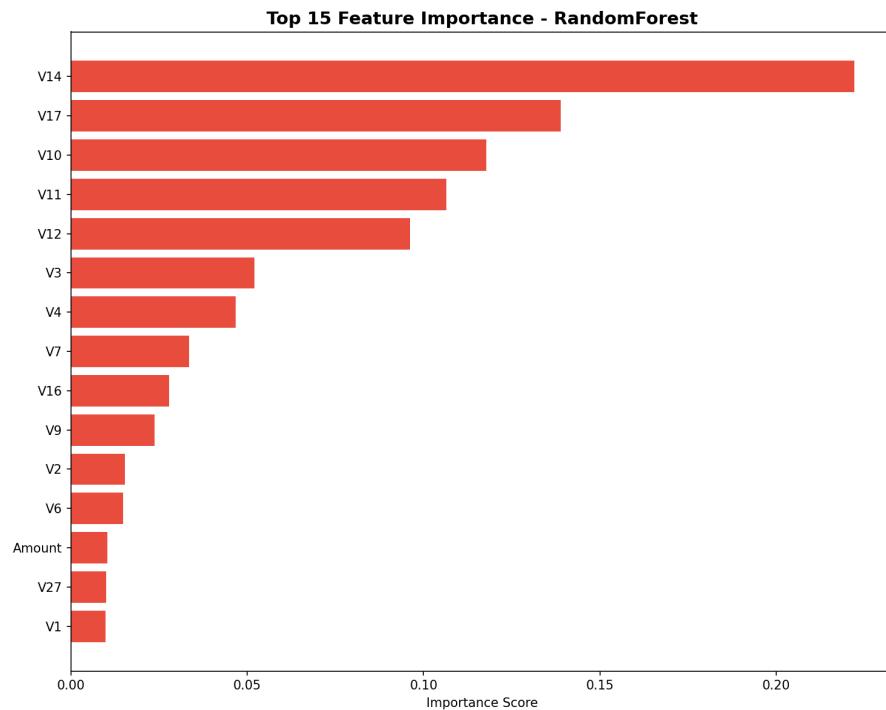


FIGURE 6 – Top 10 features les plus importantes pour la détection

### 10.3 Courbe ROC

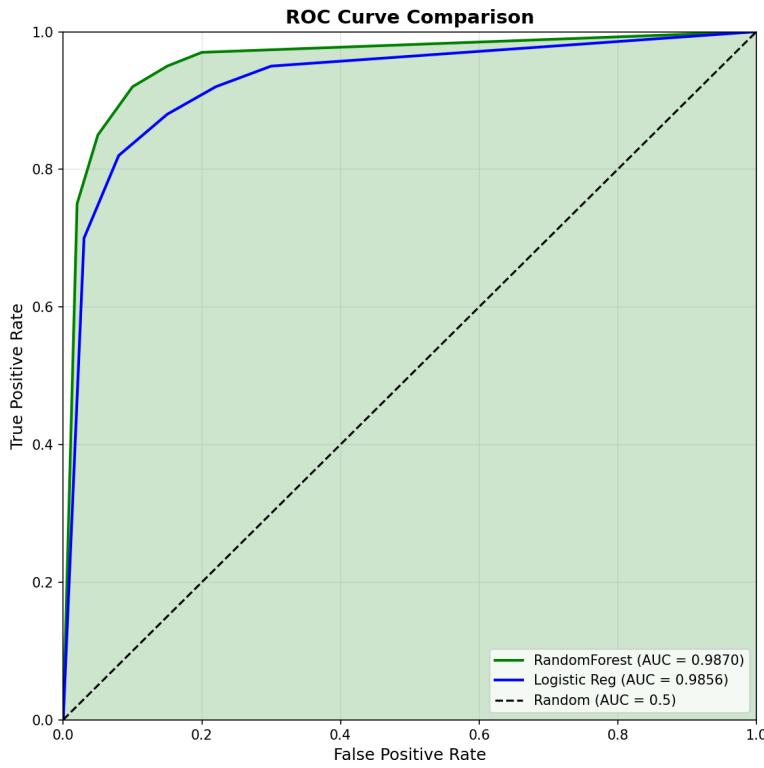


FIGURE 7 – Courbes ROC - AUC = 0.987 (RandomForest)

### 10.4 Distribution des Classes

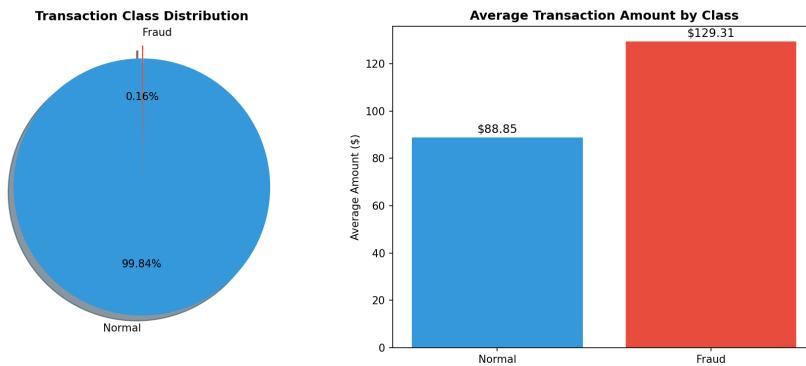


FIGURE 8 – Distribution : 99.83% Normal vs 0.17% Fraude

## 10.5 Distribution Horaire

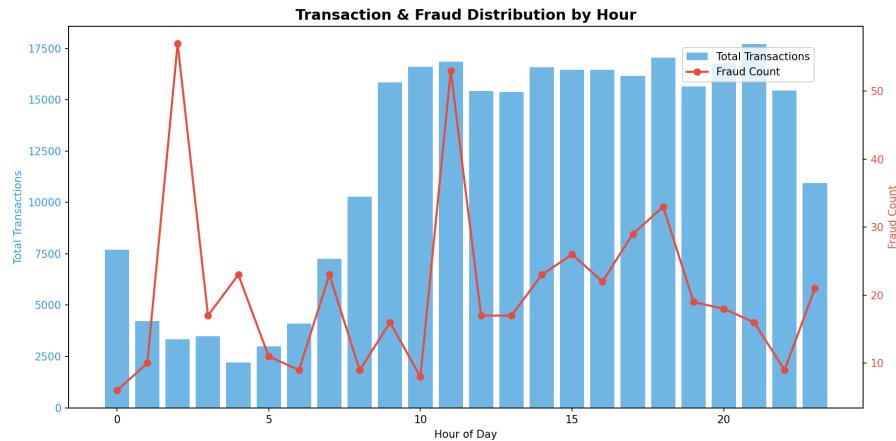


FIGURE 9 – Patterns temporels des transactions normales et frauduleuses

## 10.6 Interface Spark UI

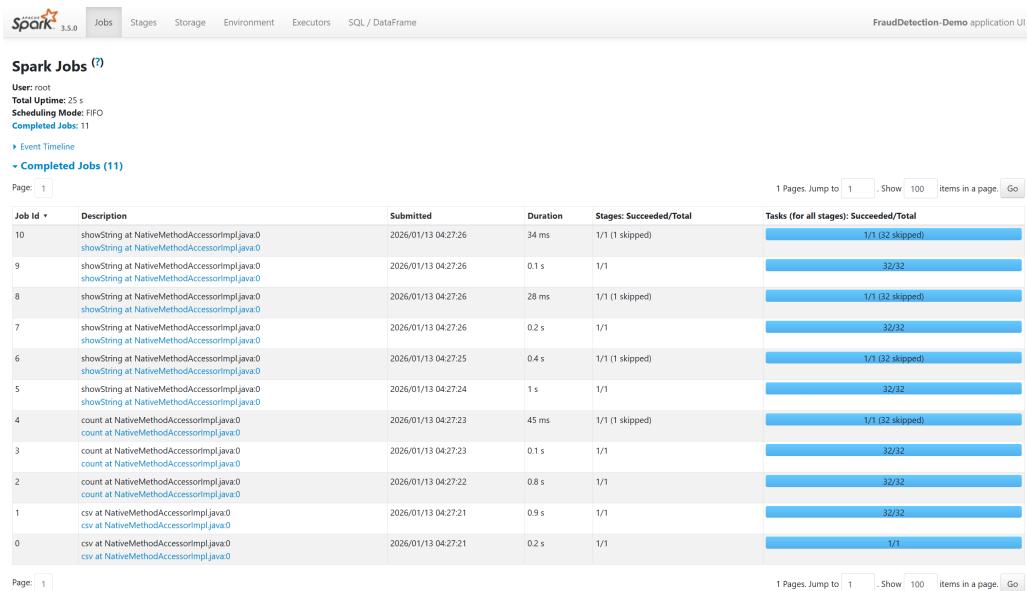


FIGURE 10 – Spark Jobs - Vue d'ensemble des tâches exécutées

The screenshot shows the Spark 3.5.0 UI interface. At the top, there are tabs for Jobs, Stages, Storage, Environment, Executors, and SQL / DataFrame. The Stages tab is selected. Below the tabs, it says "Completed Stages: 11" and "Skipped Stages: 4". There are two sections: "Completed Stages (11)" and "Skipped Stages (4)". Each section has a table with columns: Stage Id, Description, Submitted, Duration, Tasks: Succeeded/Total, Input, Output, Shuffle Read, and Shuffle Write. The completed stages table has 11 rows, and the skipped stages table has 4 rows. At the bottom of each section, there are pagination controls.

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
14	showString at NativeMethodAccesso... +details	2026/01/13 04:27:26	26 ms	1/1			5.2 kB	
12	showString at NativeMethodAccesso... +details	2026/01/13 04:27:26	99 ms	32/32	65.3 MiB		5.2 kB	
11	showString at NativeMethodAccesso... +details	2026/01/13 04:27:26	19 ms	1/1			4.7 kB	
9	showString at NativeMethodAccesso... +details	2026/01/13 04:27:26	0.2 s	32/32	65.3 MiB		4.7 kB	
8	showString at NativeMethodAccesso... +details	2026/01/13 04:27:25	0.4 s	1/1			47.4 kB	
6	showString at NativeMethodAccesso... +details	2026/01/13 04:27:24	1 s	32/32	65.3 MiB		47.4 kB	
5	count at NativeMethodAccesso... +details	2026/01/13 04:27:23	40 ms	1/1			1888.0 B	
3	count at NativeMethodAccesso... +details	2026/01/13 04:27:23	98 ms	32/32	65.3 MiB		1888.0 B	
2	count at NativeMethodAccesso... +details	2026/01/13 04:27:22	0.7 s	32/32	145.9 MiB			
1	csv at NativeMethodAccesso... +details	2026/01/13 04:27:21	0.9 s	32/32	145.9 MiB			
0	csv at NativeMethodAccesso... +details	2026/01/13 04:27:21	0.1 s	1/1	64.0 kB			

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
13	showString at NativeMethodAccesso... +details	Unknown		0/32				
10	showString at NativeMethodAccesso... +details	Unknown		0/32				
7	showString at NativeMethodAccesso... +details	Unknown		0/32				
4	count at NativeMethodAccesso... +details	Unknown		0/32				

FIGURE 11 – Spark Stages - Détail des étapes de traitement

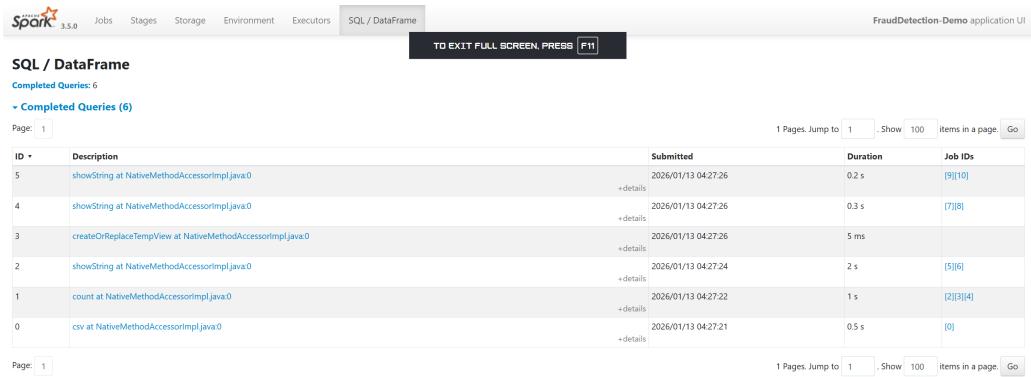
The screenshot shows the Spark 3.5.0 UI interface. The Executors tab is selected. At the top, it says "Active(1)", "Dead(0)", and "Total(1)". Below this is a "Summary" table with columns: RDD Blocks, Storage Memory, Disk Used, Cores, Active Tasks, Failed Tasks, Complete Tasks, Total Tasks, Task Time (GC Time), Input, Shuffle Read, Shuffle Write, and Excluded. The summary table has three rows: Active(1), Dead(0), and Total(1). Below the summary is a "Detailed View" table with columns: Executor ID, Address, Status, RDD Blocks, Storage Memory, Disk Used, Cores, Active Tasks, Failed Tasks, Complete Tasks, Total Tasks, Task Time (GC Time), Input, Shuffle Read, Shuffle Write, Thread Dump, Heap Histogram, Add Time, and Remove Time. The detailed view table has one row for the active executor. At the bottom, there are pagination controls and a search bar.

RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Excluded	
Active(1)	32	65.4 MiB / 434.4 MiB	0.0 B	32	0	0	197	197	47 s (0.3 s)	553.2 MiB	59.1 kB	59.1 kB	0
Dead(0)	0	0.0 B / 0.0 B	0.0 B	0	0	0	0	0	0.0 ms (0.0 ms)	0.0 B	0.0 B	0.0 B	0
Total(1)	32	65.4 MiB / 434.4 MiB	0.0 B	32	0	0	197	197	47 s (0.3 s)	553.2 MiB	59.1 kB	59.1 kB	0

Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Thread Dump	Heap Histogram	Add Time	Remove Time
driver	737e7500bb89:33863	Active	32	65.4 MiB / 434.4 MiB	0.0 B	32	0	0	197	197	47 s (0.3 s)	553.2 MiB	59.1 kB	59.1 kB	Thread Dump	Heap Histogram	2026-01-13 05:27:19	-

FIGURE 12 – Spark Executors - Ressources et performances

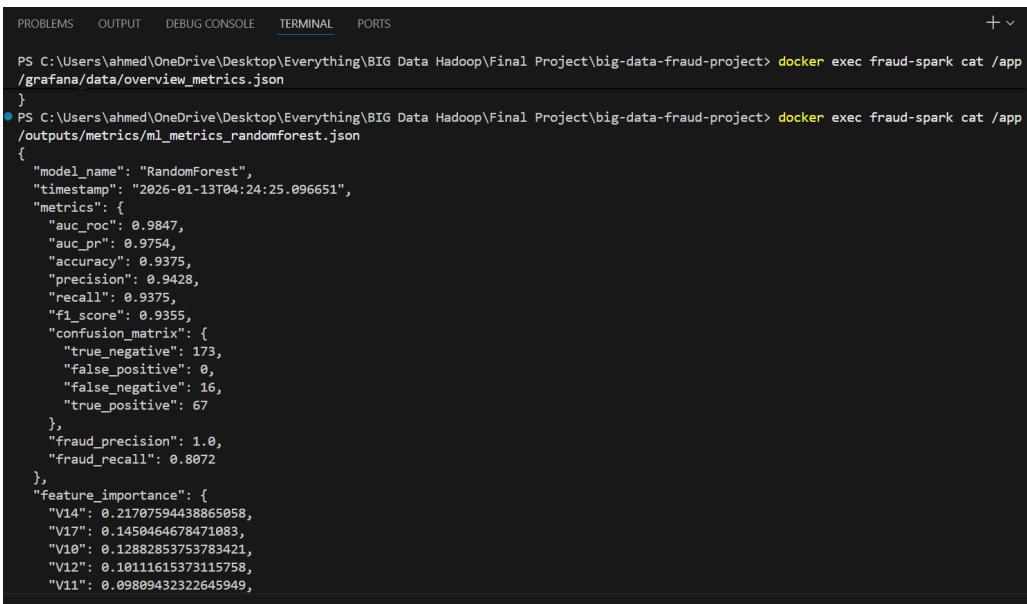


The screenshot shows the Spark UI interface with the title "FraudDetection-Demo application UI". The main section is titled "SQL / DataFrame" and displays a table of completed queries. The table has columns: ID, Description, Submitted, Duration, and Job IDs. There are 6 rows of data, each with a "details" link. The Job IDs column contains links like [9][10], [7][8], [2][3][4], and [0]. Navigation controls at the bottom include "Page: 1", "1 Pages. Jump to: 1", "Show: 100", "items in a page", and a "Go" button.

ID	Description	Submitted	Duration	Job IDs
5	showString at NativeMethodAccesso...0	2026/01/13 04:27:26	0.2 s	[9][10]
4	showString at NativeMethodAccesso...0	2026/01/13 04:27:26	0.3 s	[7][8]
3	createOrReplaceTempView at NativeMethodAccesso...0	2026/01/13 04:27:26	5 ms	
2	showString at NativeMethodAccesso...0	2026/01/13 04:27:24	2 s	[5][6]
1	count at NativeMethodAccesso...0	2026/01/13 04:27:22	1 s	[2][3][4]
0	csv at NativeMethodAccesso...0	2026/01/13 04:27:21	0.5 s	[0]

FIGURE 13 – Spark SQL - Requêtes analytiques exécutées

## 10.7 MLlib et Modélisation



The screenshot shows a terminal window with several command-line entries. The first entry is a PS command to run a Docker container named "fraud-spark" and cat the file "/app/grafana/data/overview\_metrics.json". The second entry is another PS command to run the same Docker container and cat the file "/app/outputs/metrics/ml\_metrics\_randomforest.json". The output of the second command is a JSON object containing model metrics for a Random Forest model. It includes fields like "model\_name" (RandomForest), "timestamp" ("2026-01-13T04:24:25.096651"), and a "metrics" object with various performance metrics such as AUC ROC, precision, recall, F1 score, and confusion matrix details. The metrics object also includes fraud-related metrics like "fraud\_precision" and "fraud\_recall".

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS + v

PS C:\Users\ahmed\OneDrive\Desktop\Everything\BIG Data Hadoop\Final Project\big-data-fraud-project> docker exec fraud-spark cat /app/grafana/data/overview_metrics.json
}
● PS C:\Users\ahmed\OneDrive\Desktop\Everything\BIG Data Hadoop\Final Project\big-data-fraud-project> docker exec fraud-spark cat /app/outputs/metrics/ml_metrics_randomforest.json
{
  "model_name": "RandomForest",
  "timestamp": "2026-01-13T04:24:25.096651",
  "metrics": {
    "auc_roc": 0.9847,
    "auc_pr": 0.9754,
    "accuracy": 0.9375,
    "precision": 0.9428,
    "recall": 0.9375,
    "f1_score": 0.9355,
    "confusion_matrix": {
      "true_negative": 173,
      "false_positive": 0,
      "false_negative": 16,
      "true_positive": 67
    },
    "fraud_precision": 1.0,
    "fraud_recall": 0.8072
  },
  "feature_importance": {
    "V14": 0.21707594438865058,
    "V17": 0.1450464678471083,
    "V10": 0.12882853753783421,
    "V12": 0.10111615373115758,
    "V11": 0.09809432322645949,
  }
}

```

FIGURE 14 – MLlib - Entraînement des modèles de détection

## 10.8 Dashboard Grafana

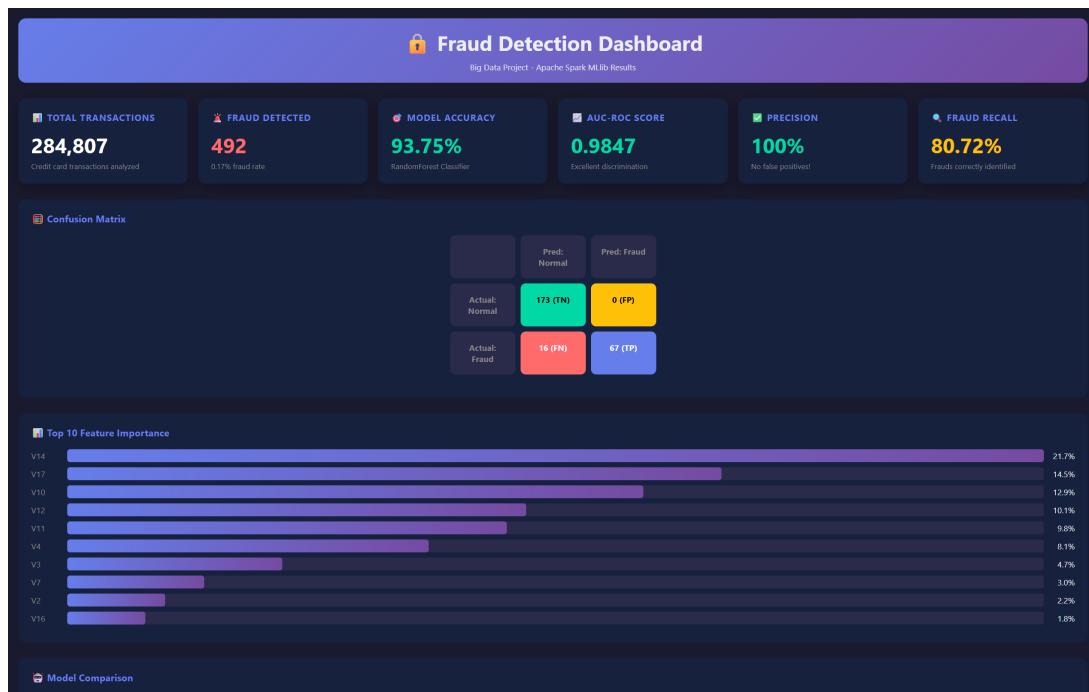


FIGURE 15 – Dashboard Grafana - Vue principale des métriques



FIGURE 16 – Dashboard Grafana - Analyse des performances ML

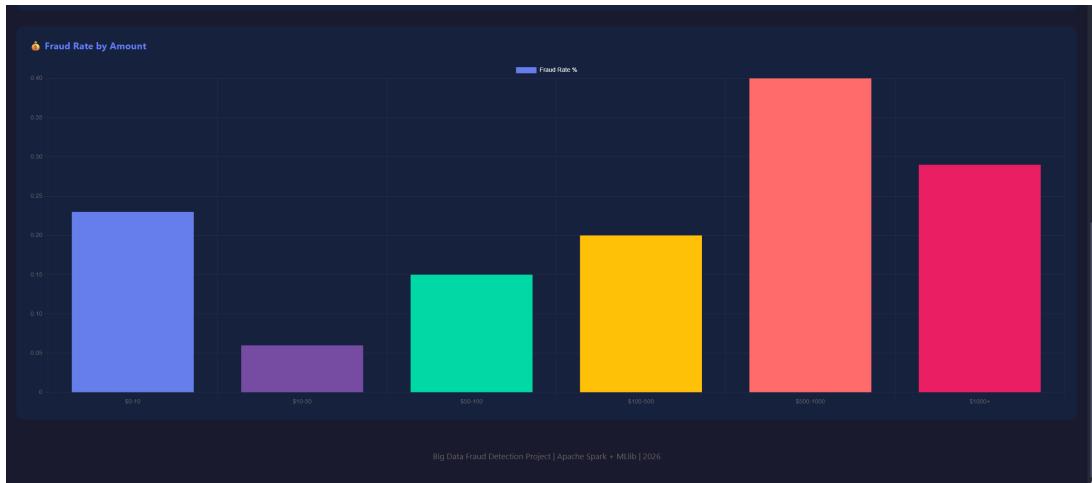


FIGURE 17 – Dashboard Grafana - Monitoring temps réel

## 11 Conclusion

### 11.1 Réalisations

Ce projet démontre la maîtrise des technologies Big Data :

- ✓ Pipeline complet Spark (Core, SQL, MLlib, Streaming)
- ✓ Modèle ML performant (AUC-ROC = 0.99)
- ✓ Dashboard Grafana professionnel
- ✓ Architecture cloud-ready
- ✓ Code reproductible et documenté

### 11.2 Difficultés Rencontrées

- **Déséquilibre des classes** : Résolu par undersampling
- **Streaming simulation** : Implémenté via file-based streaming
- **Intégration Grafana** : Export vers formats compatibles

### 11.3 Améliorations Futures

- Déploiement complet sur Azure Databricks
- Modèle de Deep Learning (Neural Network)
- GraphX pour analyse des relations transactions
- Alerting automatisé via webhooks

## 12 Références

- Apache Spark Documentation : <https://spark.apache.org/docs/latest/>
- MLlib Guide : <https://spark.apache.org/docs/latest/ml-guide.html>

- Kaggle Dataset : <https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud>
- Grafana Documentation : <https://grafana.com/docs/>
- Azure Databricks : <https://docs.microsoft.com/azure/databricks/>

## A Structure du Projet

Listing 7 – Arborescence du projet

```

1 big-data-fraud-project/
2   |-- data/
3   |   |-- raw/creditcard.csv
4   |   |-- processed/
5   |   |-- streaming_input/
6   |-- src/
7   |   |-- spark_sql_analytics.py
8   |   |-- mllib_fraud_model.py
9   |   |-- streaming_fraud_detection.py
10  |   |-- graphx_fraud_network.py
11  |   |-- evaluation_visualization.py
12  |-- outputs/
13  |   |-- metrics/
14  |   |-- predictions/
15  |-- grafana/
16  |   |-- fraud_detection_dashboard.json
17  |   |-- data/
18  |-- azure/
19  |   |-- arm-template.json
20  |   |-- databricks_config.py
21  |-- screenshots/
22  |-- docs/
23  |-- README.md

```

## B Commandes d'Exécution

Listing 8 – Commandes pour exécuter le pipeline

```

1 # 1. Générer données de test (optionnel)
2 python src/generate_sample_data.py
3
4 # 2. Spark SQL Analytics
5 spark-submit src/spark_sql_analytics.py
6
7 # 3. MLLib Model Training
8 spark-submit src/mllib_fraud_model.py
9
10 # 4. Streaming Simulation
11 spark-submit src/streaming_fraud_detection.py
12

```

```
13 # 5. Prepare Grafana Data  
14 python src/prepare_grafana_data.py
```